Symbolic Pattern Matching and Rule-Based Programming Paradigm

Tošić, V., Dejan

***Abstract*—Present-day computer algebra systems offer programming paradigms that can solve many scientific and engineering problems more efficiently, with less effort from the programmer, than the classic programming styles, such as procedural or object-oriented programming. This contribution focuses on a real-life engineering problem that is nicely solved by using symbolic pattern matching and rule-based programming. The benefits of the new approach are highlighted and the entire code in Mathematica is presented and explained. The symbolic programming language is suggested for rapid software development and algorithm prototyping in the field of scientific research and engineering practice.**

Index Terms—*computer algebra system, Foster's theorem, Mathematica, symbolic pattern matching, rule-based programming*

1. Introduction

A common approach taken in modern software engineering is to combine various programming paradigms, such as procedural programming or object-oriented programming, to achieve the desired goals. Since the appearance of computer algebra systems (CAS), programming has become a task of knowledge accumulation that tells the computer what to know, when to use information, and how to apply the knowledge in solving problems. Contemporary CAS integrated a suite of software tools and programming concepts that are particularly useful for engineers and physicists. The leader in implementing CAS, *Mathematica* by Wolfram Research, launched an original programming language, which gives a different perspective to software realization of scientific and engineering algorithms [1,2,3].

2. Mathematica Pattern Matching

At the core of *Mathematica* are its highly developed symbolic language and the foundational idea that everything – data, programs, mathematical formulas, lists, graphics, and documents – can be represented as symbolic expressions. The advanced notion of expressions is a crucial unifying principle and it is the fact that every object has the same underlying structure.

Expressions are used to specify operations and to maintain a structure, which can then be acted on by the operations. A prototypical example of an expression is `f[x,y]`, where the symbol `f` is the head of the expression, the symbols `x` and `y` are the arguments, and the square brackets are delimiters; the head and the arguments itself can be expressions, as well. The parentheses are used exclusively for grouping following standard mathematical notation to specify the precedence of operators. The use of distinct delimiters for arguments is a unique concept important for symbolic programming and it adds a new level of flexibility to the very concept of programming: the pattern matching and transformation rules can be applied to both expression heads and expression arguments.

Patterns are used to represent classes of expressions with a given structure. The main power of patterns comes from the fact that many operations can be done not only with single expressions, but also with patterns that represent whole classes of expressions.

It might be useful to mention that (1) a pattern will match a particular expression if the structure of the pattern is the same as the structure of the expression, (2) even though two expressions may be mathematically equal, they cannot be represented by the same pattern unless they have the same structure.

The fact that patterns specify the structure of expressions is crucial in making it possible to set up transformation rules which change the structure of expressions, while leaving them mathematically equal [4,5,6].

3. Application to Electrical Engineering

Symbolic pattern matching and rule-based programming (and the combining of various programming paradigms) are actually explored in the AI community for many years, but are not widespread in the engineering community. Therefore, the following example illustrates this paradigm from the electrical engineering viewpoint and demonstrates the uniqueness and benefits of the symbolic language concept.

Consider an essential problem of electrical network synthesis and practical filter design [7,8]:

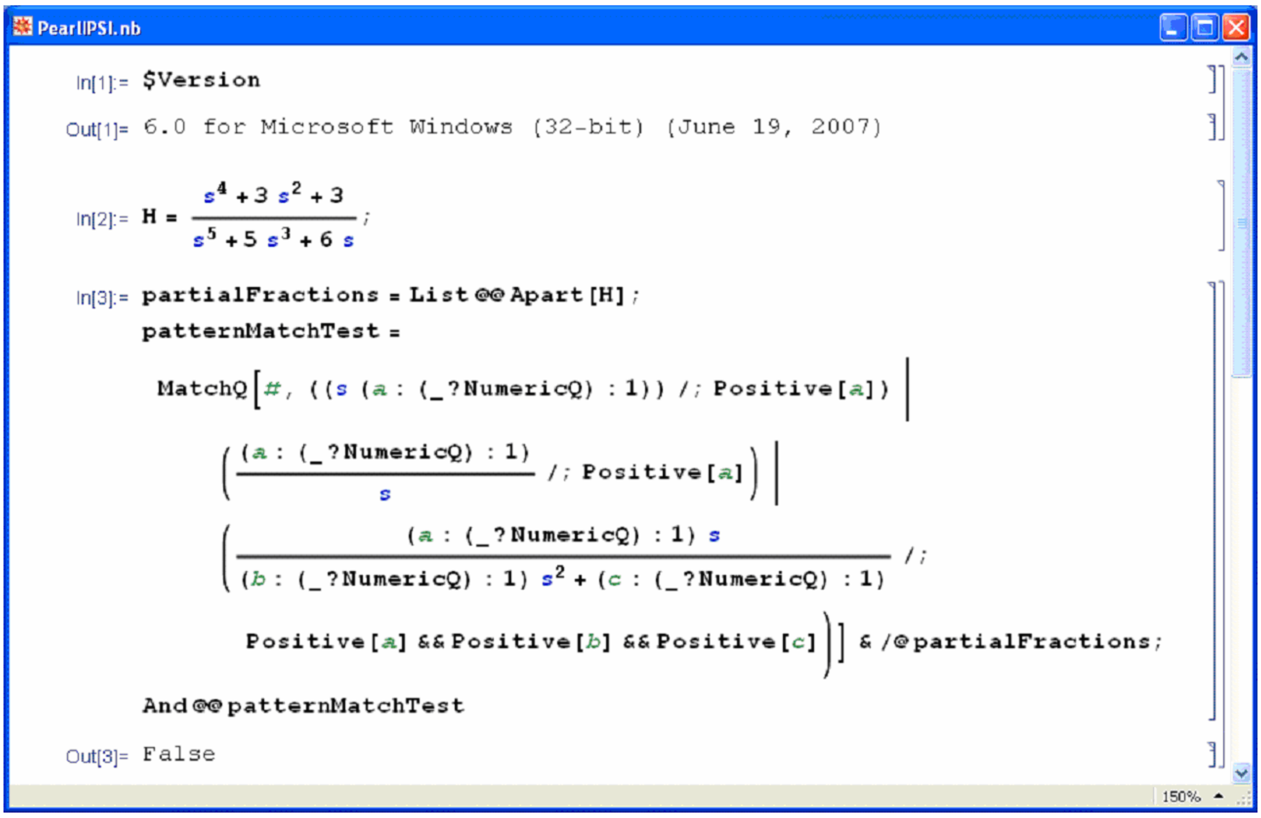
Manuscript received March 30, 2007. This work was supported in part by the Ministry of Science and Environmental Protection of the Republic of Serbia under grant no. TR-6154.

D. V. Tošić is with the School of Electrical Engineering, University of Belgrade, Serbia (e-mail: tosic@etf.bg.ac.yu).

given a transfer function, e.g. $H = \frac{s^4 + 3s^2 + 3}{s^5 + 5s^3 + 6s}$, determine whether the function can be realized as a driving-point impedance of an electrical network of interconnected capacitors and inductors.

According to the Foster's reactance theorem, an algebraic rational function to be realizable as the driving-point impedance of a lossless one-port electrical network can always be expanded as $H = \frac{a_{-1}}{s} + a_0 s + \sum_i \frac{a_i s}{b_i s^2 + c_i}$, where all coefficients are positive. *Mathematica* code that performs the required test, based on the Foster's theorem, is given in Fig. 1. The code is concise, elegant, readable, easy-to-maintain, and self-explanatory.

The function **`Apart`** expands the transfer function (**`H`**) into partial fractions and the function **`List`** converts the expansion to a list of terms (**`partialFractions`**). The function **`MatchQ`** performs the desired pattern matching, term by term, under the conditional rule that the coefficients should be positive numbers; it returns a list of logical constants (**`patternMatchTest`**). The function **`And`** returns true if all terms pass the pattern match, otherwise it returns false. The intermediate results are shown in Fig. 2.



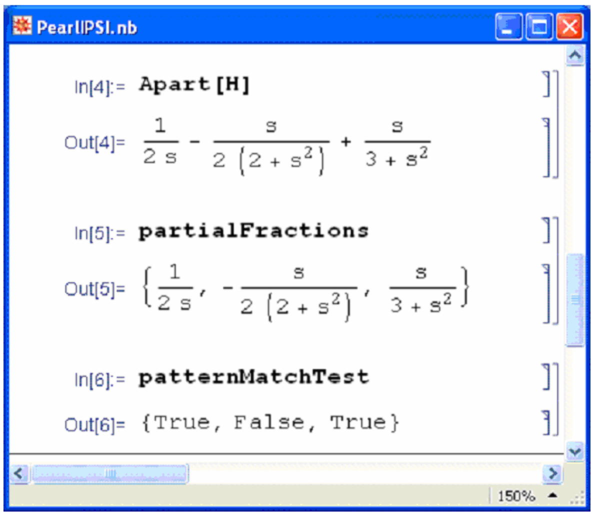
```
In[1]:= $Version
Out[1]= 6.0 for Microsoft Windows (32-bit) (June 19, 2007)

In[2]:= H = (s^4 + 3 s^2 + 3)/(s^5 + 5 s^3 + 6 s);

In[3]:= partialFractions = List @@ Apart[H];
        patternMatchTest =
          MatchQ[#, ((s (a : (_?NumericQ) : 1)) /; Positive[a]) |
             ((a : (_?NumericQ) : 1)/s /; Positive[a]) |
             ((a : (_?NumericQ) : 1) s)/((b : (_?NumericQ) : 1) s^2 + (c : (_?NumericQ) : 1)) /;
               Positive[a] && Positive[b] && Positive[c])] & /@ partialFractions;

        And @@ patternMatchTest
Out[3]= False
```

Figure 1: *Mathematica* code that performs the test based on the Foster's theorem.



```
In[4]:= Apart[H]
Out[4]= 1/(2 s) - s/(2 (2 + s^2)) + s/(3 + s^2)

In[5]:= partialFractions
Out[5]= {1/(2 s), -s/(2 (2 + s^2)), s/(3 + s^2)}

In[6]:= patternMatchTest
Out[6]= {True, False, True}
```

Figure 2: Results of the test shown in Fig. 1.

4. CONCLUSION

Symbolic pattern matching and rule-based programming paradigm is an important issue and a choice of preference for rapid software development and algorithm prototyping in the fields of science and engineering. It is the key programming paradigm involved in the development and implementation of *SchematicSolver* [9].

REFERENCES

[1] Tošić, D., Lutovac, M., "Advances in symbolic simulation of systems," *The IPSI BgD Transactions on Advanced Research*, vol. 3, no. 1, 2007, pp. 9–14.

[2] Wolfram, S., *The Mathematica Book*. Cambridge, MA: Cambridge University Press, Wolfram Media, 2003.

[3] *Mathematica*, http://www.wolfram.com/, Version 6.0.1 released June 19, 2007.

[4] Maeder, R., *Computer Science with Mathematica*. New York, NY: Cambridge University Press, 2000.

[5] Bahder, T., *Mathematica for Scientists and Engineers*. Reading, MA: Addison-Wesley, 1995.

[6] Blachman, N., *Mathematica: A Practical Approach*. Englewood-Cliffs, NJ: Prentice Hall, 1992.

[7] Chen, W-K. (Ed.), *The Electrical Engineering Handbook*. Burlington, MA: Elsevier Academic Press, 2004.

[8] Lutovac, M. D., Tošić, D. V., Evans, B. L., *Filter Design for Signal Processing using MATLAB and Mathematica*. Upper Saddle River, NJ: Prentice Hall, 2001.

[9] Lutovac, M. D., Tošić, D. V., *SchematicSolver*, http://www.wolfram.com/products/applications/schematicsolver/, Version 2.1 for *Mathematica* 6 rel. July 12, 2007.